Introduction au logiciel Matlab

Marie Postel*

Version révisée septembre 2004

Table des matières

*Laboratoire Jacques-Louis Lions, Université Pierre et Marie Curie

Introduction

MATLAB[1] est un logiciel commercial de calcul interactif. Il permet de réaliser des simulations numériques basées sur des algorithmes d'analyse numérique. Il peut donc être utilisé pour la résolution approchée d'équations différentielles, d'équations aux dérivées partielles ou de systèmes linéaires, etc... L'objectif de ces séances `Matlab` est double : la connaissance de ce logiciel est en soi indispensable parce qu'il est de plus en plus utilisé dans l'industrie et les banques pour développer des prototypes de logiciels et tester de nouveaux algorithmes. Tous les ans entre vingt et trente pour cent des étudiants du DESS utilisent `Matlab` pendant leur stage.
Ensuite son apprentissage va passer par la mise en pratique des algorithmes d'analyse numérique étudiés plus théoriquement dans le reste du module. Pour cela on réalisera au cours des séances un projet directement inspiré d'un cas concret schématisant un problème industriel.
Signalons au passage que des logiciels en shareware/freeware émulent `Matlab` de manière de plus en plus satisfaisante. Leur utilisation permet de palier l'inconvénient principal du coût de la licence. Vous pouvez par exemple télécharger `Scilab` gratuitement sur le site Internet de l'INRIA[2]. Vous pouvez apprendre à programmer en `Scilab` dont la syntaxe est proche de celle de `Matlab` avec l'ouvrage récent et complet de G. Allaire et S. M. Kaber[3][3, 2]

1 Prise en main de `Matlab`

Version imprimable du document

Cliquer ici[4] pour télécharger la version imprimable de ce document en format Postscript.
Cliquer ici[5] pour télécharger la version imprimable de ce document en format PDF.

Fonctionnement général

Ce document est destiné à être utilisé sur un poste de travail LINUX ou UNIX, typiquement les ordinateurs de la salle 1C22 à Chevaleret. Ouvrez simultanément une session `Matlab` en tapant tout simplement
`Matlab`
dans une fenêtre de commandes.
Vous allez voir apparaître une ou plusieurs nouvelles fenêtres sur votre écran, dont la *fenêtre de commandes* représentée sur la figure 1. Les principales caractéristiques de cette fenêtre sont :
Les caractères >> en début de ligne constituent le *prompt* de `Matlab`. C'est après eux que vous pouvez taper des commandes qui seront exécutées par le logiciel après avoir tapé sur la touche *entrée*.
Le résultat de l'exécution s'inscrit alors dans la fenêtre ou est représenté graphiquement dans une nouvelle fenêtre spécifique (avec possibilité de zoom, d'impression, etc...). Pour rentrer une suite complexe d'instructions (on parle aussi d'un script), on les tape au préalable dans un fichier en utilisant l'éditeur intégré. Une fois le script enregistré, on peut l'exécuter en tapant son nom dans la fenêtre `Matlab` (cf. exercice 1).
L'historique des instructions entrées depuis le début de la session sur la ligne de commande est accessible par pressions successives de la touche ↑.
Enfin, pour effacer les données en mémoire (par exemple avant d'exécuter un nouveau calcul), il suffit d'utiliser la commande `clear`.
Exemple :
Tapez dans la fenêtre de commande la ligne suivante
>>A = ones(2,3)

---
[1] `http://www.mathworks.com`
[2] `http://www-rocq.inria.fr/scilab`
[3] `http://www.ann.jussieu.fr/algebrelineairenumerique/`
[4] `http://www.ann.jussieu.fr/~postel/dess/MATLAB.ps`
[5] `http://www.ann.jussieu.fr/~postel/dess/MATLAB.pdf`

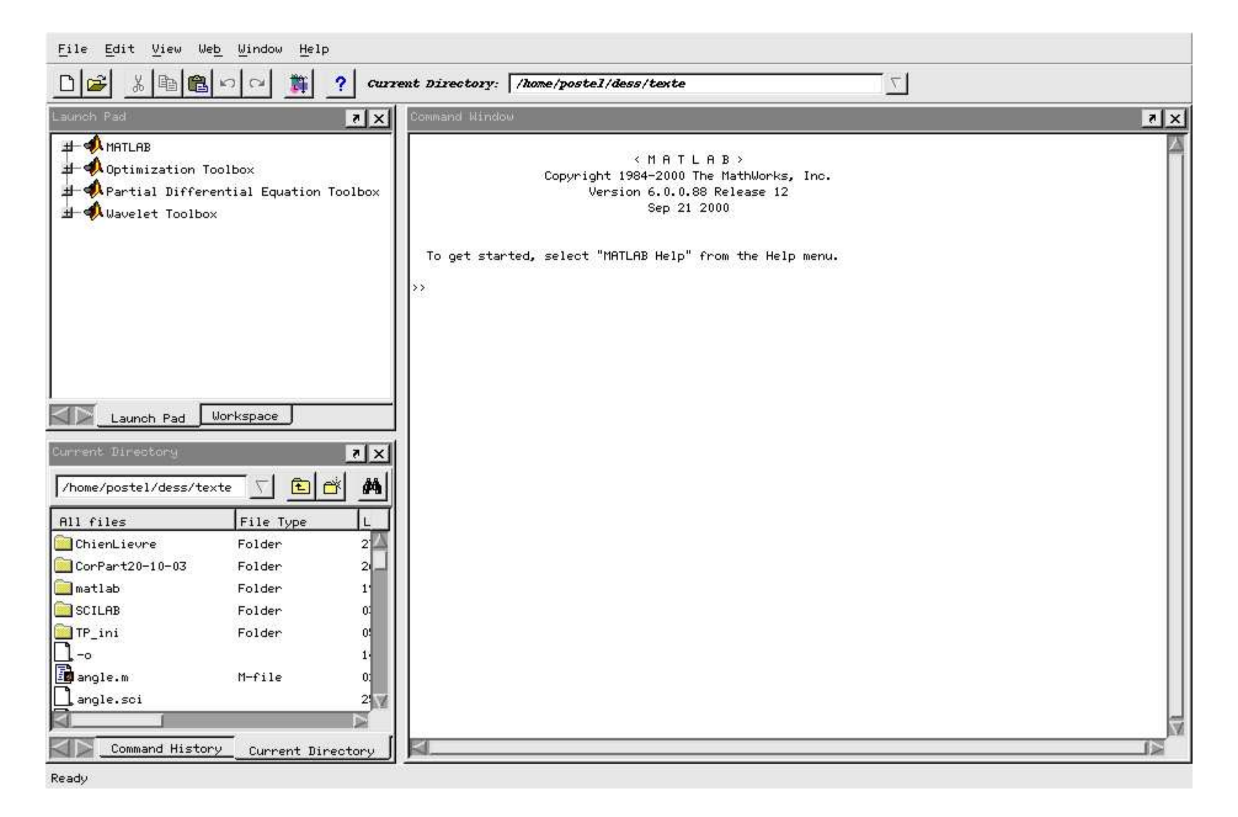


FIG. 1 – Fenêtre de commandes `Matlab`

Le résultat devrait ressembler à :

```
A = ones(2,3)

        1         1         1
        1         1         1
```

Ceci représente une matrice 2×3 dont toutes les composantes valent un. `Matlab` est assez friand de lignes blanches. Si vous préférez des sorties plus compactes tapez
>> format compact
en début de session.
Si vous ne désirez pas voir le résultat d'une commande immédiatement, il faut la faire suivre d'un point virgule. La commande sera exécutée, mais son résultat n'apparaîtra pas à l'écran. Essayez la séquence de commandes suivantes :
>>clear
>>A
>>A = ones(2,3) ;
>>A

Enfin si vous voulez que `Matlab` ignore complètement ce que vous tapez - ce qui sera utile dans l'écriture des scripts pour y mettre des commentaires améliorant leur lisibilité, il faut le faire précéder du caractère %.

Utilisation de l'aide en ligne

L'apprentissage du bon maniement de l'aide (en ligne ou papier) du logiciel est indispensable : étant donné le très grand nombre d'instructions utilisables, il est hors de question de pouvoir mémoriser chacune d'elles avec sa syntaxe correspondante. À noter également que cette aide est uniquement disponible en langue anglaise (ce qui nécessite le cas échéant l'apprentissage complémentaire de quelques rudiments de cette langue...).
L'aide en ligne permet de retrouver toutes les informations utiles : en allant dans le menu `Help->MATLAB help`, (voir figure 1) une nouvelle fenêtre s'ouvre partagée en deux. A gauche, on peut en cliquant sur le signet supérieur correspondant, activer

- `Contents`. La table des matières de l'aide en ligne. En cliquant sur un chapitre de l'arborescence, son contenu est affiché dans la fenêtre de droite.
- `Index`. L'index de l'aide en ligne, qui répertorie toutes les commandes `Matlab` et permet d'accéder à leur mode d'emploi spécifique. On peut taper le nom de la commande - si on la connaît !- dans la fenêtre de saisie `Search index for :`, ou bien la rechercher dans la liste alphabétique proposée. Pour avancer dans l'alphabet taper la première lettre de la commande recherchée dans la fenêtre de saisie.
- `Search`. Un moteur de recherche dans l'index
- `Favorites`. La liste des chapitres de l'aide stockés dans les favoris. Pour ajouter un chapitre dans cette liste, se positionner dessus à partir de *Contents* et cliquer sur le bouton de droite de la souris.

Une aide très importante se présente sous la forme de programmes de démonstration auxquels on peut accéder à partir de *Contents* $->$ *Begin Here* puis en cliquant sur *demos* dans la sous-fenêtre de droite, ou bien en tapant *demos* directement dans la fenêtre de commande.

2 Programmation

2.1 Syntaxe du langage

Un script `Matlab` est composé d'une suite d'instructions, toutes séparées par une virgule (ou de manière équivalente, un passage à la ligne) ou un point virgule. La différence entre ces deux types de séparation est liée à l'affichage ou non du résultat à l'écran (seulement effectué dans le premier cas). Comme tout langage, `Matlab` possède aussi un certain nombre d'instructions syntaxiques (boucles simples, conditionnelles, etc...) et de commandes élémentaires (lecture, écriture, etc...) récapitulées dans les tableaux 2-3.
Dès que le calcul à effectuer implique un enchaînement de commandes un peu compliqué, il vaut mieux écrire ces dernières dans un fichier. Par convention un fichier contenant des commandes `Matlab` porte un nom avec le suffixe *.m* et s'appelle pour cette raison un *M-file* ou encore *script*. On utilisera TOUJOURS l'éditeur intégré au logiciel qui se lance à partir de la fenêtre de commande en cliquant sur les icônes `new M-file` ou `open file` dans la barre de menu. Une fois le fichier enregistré sous un nom valide, on peut exécuter les commandes qu'il contient en tapant son nom - sans le suffixe .m - dans la fenêtre de commande. Si vous avez ouvert l'éditeur comme indiqué, à partir de la fenêtre de commande, les `M-file` seront créés dans le répertoire courant, accessible depuis cette fenêtre, et vous n'aurez pas de problème d'accès. Si vous voulez exécuter des scripts qui se trouvent ailleurs dans l'arborescence des fichiers, vous aurez éventuellement à modifier le *Path* en cliquant sur le menu `file`$-$ `>SetPath` ou bien en changeant de répertoire de travail (cliquer sur l'onglet `current directory`).

Exercice 1 Un premier script
Créer dans le répertoire courant un *M-file* en cliquant sur l'icone `New M-file` et taper les instructions suivantes :

Script 1 *Premier script*

```
a=1;
b=2;
c=a+b;
```

Sauver (en cliquant sur l'icone `save`) sous le nom `PremierScript.m` et exécuter la commande
>>PremierScript
dans la fenêtre `Matlab`, soit en la tapant au clavier soit en cliquant sur l'icone `Run` à partir de la fenêtre d'édition. Taper maintenant
>>c
qui doit contenir la valeur calculée par le script.

Plusieurs types de données sont disponibles dans `Matlab`. Les types traditionnels que l'on retrouve dans tous les langages de programmation : les types numériques (*single*, *double*, *int8*, etc...), caractères *char* ,les tableaux de réels, et les tableaux creux *sparse*, et les types composés *cell*, *structure* ainsi que les types définis par l'utilisateur, comme les fonctions *inline*. Le type de donnée privilégié sous `Matlab` est les **tableaux** à une ou deux dimensions, qui correspondent aux vecteurs et matrices utilisés en mathématiques et qui sont aussi utilisés pour la représentation graphique. Nous allons donc nous attarder sur leur définition et leur maniement dans les paragraphes qui suivent.

2.2 Vecteurs

Pour définir un vecteur la syntaxe est une des suivantes :

```
>>v=[2;3;7]                 %vecteur colonne composantes réelles
v=
   2.0000
   3.0000
   7.0000
>>v=[2,-3+i,7]   %vecteur ligne composantes complexes, i^2 =-1
v =
   2.0000            -3.0000 + 1.0000i   7.0000
>>v'                               % vecteur transconjugué
ans =
   2.0000
  -3.0000 - 1.0000i
   7.0000
>>v.'                               % vecteur transposé
ans =
   2.0000
  -3.0000 + 1.0000i
   7.0000
>>w=[-3;-3-i;2]                % vecteur colonne
w =
  -3.0000
  -3.0000 - 1.0000i
   2.0000
>>v+w'                        % somme de deux vecteurs
ans =
  -1.0000            -6.0000 + 2.0000i   9.0000
>>v*w                          % produit scalaire euclidien
 ans  =
    18.
>>w'.*v    % produit des composantes terme à terme
ans =
  -6.0000             8.0000 - 6.0000i  14.0000
```

```
>>w'.*v    % division des composantes terme à terme
>>w.^3     % mise à la puissance 3 de chaque composante.
```

Les composantes sont séparées par des blancs (dangereux) ou de préférence par des virgules pour les vecteurs lignes et par des points-virgules pour les vecteurs colonnes. Des messages erreurs sont affichés si une opération impossible est tentée (par exemple l'addition de vecteurs de longueurs différentes).
Et pour aller plus vite...

```
>>v=1:5.1:23            % vecteur a increment constant
v =
    1.0000    6.1000   11.2000   16.3000   21.4000
>>ones(size(v)) % vecteur de meme taille que v et contenant des 1
 ans  =
   1  1  1  1  1
>>ones(1,4) % vecteur ligne 4 composantes egales a 1
 ans  =
   1    1    1   1
>>3*ones(1,5)  % vecteur ligne 5 composantes egales a 3
 ans  =
   3    3    3    3    3
>>zeros(3,1)  % vecteur colonne 3 composantes nulles
 ans  =
    0
    0
    0
```

2.3 Matrices

Les matrices suivent la même syntaxe que les vecteurs. Les composantes des lignes sont séparées par des virgules et chaque ligne est séparée de l'autre par un point virgule.

```
>>% une manière de définir une matrice 3 x 3:
>>A=[1,2,3;0,0,atan(1);5,9,-1];

>>% une autre syntaxe pour faire la même chose
>>A=[1 2 3
>>     0 0 atan(1)
>>     5 9 -1]
 A  =
   1.    2.    3.
   0.    0.     .7853982
   5.    9.  - 1.

>>% à ne pas confondre avec ce groupe d'instructions
>>A=[1 2 3...
>>0 0 atan(1)...
>>5 9 -1]
 A  =
   1.    2.    30.    0.    0.7853982    5.    9.  - 1.


>>v=1:5;W=v'*v            % multiplication de matrices
W =
```

```
     1     2     3     4     5
     2     4     6     8    10
     3     6     9    12    15
     4     8    12    16    20
     5    10    15    20    25
>>W(1,:)                 %extraction de  la première ligne
 ans  =
   1.    2.    3.    4.    5.
>>A=eye(3,3)     % Matrice identité
A =
     1     0     0
     0     1     0
     0     0     1
>>B=toeplitz([2,1,0,0])   % une matrice de Toeplitz
B =
     2     1     0     0
     1     2     1     0
     0     1     2     1
     0     0     1     2
```

Le tableau 1 résume les principales fonctions affectant ou effectuant des opérations sur des matrices. Noter que les fonctions scalaires courantes, (sin, exp, etc...) peuvent aussi s'appliquer à des matrices, composante par composante, comme dans l'exemple suivant

```
>>u=[0:1:4]
u =
     0     1     2     3     4
>>v=sin(u)
v =
         0    0.8415    0.9093    0.1411   -0.7568
```

La fonction $find(C(A))$ renvoie les indices dans le tableau A des composantes vérifiant la condition $C(A)$. Par exemple

```
>>A=rand(1,5)    % cree un vecteur ligne contenant 5 nombres repartis
                                % aleatoirement entre 0 et 1
>>find(A>0.5)    % renvoie les indices des composantes de A >0.5
>>find(A)         % renvoie les indices des composantes de A differentes de 0
>>find(A==0.2)    % renvoie les indices des composantes de A egales a 0.2
```

Exercice 2 Tester et comprendre les deux lignes suivantes

```
A=[1  2  -1 1 ; -1 1 0 3]
find(A>0)
```

2.4 Exercices sur la syntaxe de base et les tableaux

La solution des exercices ne doit pas être tapée directement dans la fenêtre de commandes mais au contraire dans la fenêtre de l'éditeur de manière à être sauvegardée dans un script pour pouvoir y revenir ultérieurement(cf. 1). Les mots *clefs* en *italiques* dans la marge sont des fonctions `Matlab` à utiliser pour faire l'exercice. Taper par exemple

Fonction	Description
ones(i,j)	crée un tableau de i lignes j colonnes contenant des 1
zeros(i,j)	crée un tableau de i lignes j colonnes contenant des 0
eye(i,j)	crée un tableau de i lignes j colonnes avec des 1 sur la diagonale principale et 0 ailleurs
toeplitz(u)	crée une matrice de Toeplitz symétrique dont la première ligne est le vecteur u
diag(u)	crée une matrice carrée avec le vecteur u sur la diagonale et 0 ailleurs
diag(U)	extrait la diagonale de la matrice U
triu(A)	renvoie la partie supérieure de A
tril(A)	renvoie la partie inférieure de A
linspace(a,b,n)	crée un vecteur de n composantes uniformément réparties de a à b
A\b	résolution du système linéaire `Ax=b`
cond(A)	conditionnement d'une matrice (norme euclidienne)
det(A)	déterminant d'une matrice
rank(A)	rang d'une matrice
inv(A)	inverse d'une matrice
pinv(A)	pseudo inverse d'une matrice
svd(A)	valeurs singulières d'une matrice
norm(A)	norme matricielle ou vectorielle
u'	prend le transposé de u
u*v	multiplication matricielle
u+v	addition matricielle
u-v	soustraction matricielle
u.* v	multiplication des tableaux u et v terme à terme
u./v	division du tableau u par le tableau v terme à terme
find(C(A))	indices des composantes du tableau A vérifiant la condition C(A)

TAB. 1 – Principales opérations sur les matrices.

```
help  norm
```

dans la fenêtre de commandes pour connaître la syntaxe d'utilisation de la fonction `norm`.

Exercice 3 On note u, v et w les vecteurs suivants

$$u = (1, -1, 2)^T,\ v = (10, -1, 3)^T\ w = (5, -1, 4)^T.$$

1. Calculer $3u$, $\|u\|_2$, $2u - v + 5w$, $\|2u - v + 5w\|_1$, $\|w - 4v\|_\infty$. *norm*
2. Déterminer l'angle formé par les vecteurs v et w. *acos*

Exercice 4 On note u et v les nombres complexes

$$u = 11 - 7i \quad v = -1 + 3i.$$

Calculer les modules de u et de v, les produits $u\bar{v}, v\bar{u}$, la partie réelle et la partie imaginaire de $u^3 + v^2$. *conj* *real* *imag*

Exercice 5 On note A, B et C les matrices suivantes

$$A = \begin{pmatrix} 1 & 3 & 2 \\ -5 & 3 & 1 \\ -10 & 0 & 3 \\ 1 & 0 & -2 \end{pmatrix},\ B = \begin{pmatrix} 1 & -2 & 5 \\ 6 & 1 & -1 \end{pmatrix},\ C = \begin{pmatrix} 10 & -5 \\ 3 & 1 \end{pmatrix}.$$

1. Calculer les matrices AB, BA et AB^T.

2. Calculer les matrices $D = I_2 - BB^T$. *eye*
3. Calculer les déterminants des matrices A, B, C, D et $E = AA^T$. *det*
4. Calculer les inverses des matrices A, B, C, D et $E = AA^T$. *inv*
5. Calculer les valeurs propres de la matrice E. Quel est le rayon spectral de E ? *spec*
6. Déterminer les vecteurs propres de la matrice A. *bdiag*

Exercice 6 On pose

$$A = \begin{pmatrix} 1 & -1 & 7 \\ -4 & 2 & 11 \\ 8 & 0 & 3 \end{pmatrix}, \; B = \begin{pmatrix} 3 & -2 & -1 \\ 7 & 8 & 6 \\ 5 & 1 & 3 \end{pmatrix}.$$

Que font les instructions suivantes

```
3*A; A.*B; A./B; cos(A); exp(B);
```

cos, exp
expm

Exercice 7 Pour chacune des matrices

$$A_1 = \begin{pmatrix} 1 & 2 & 3 \\ 3 & 2 & 1 \\ 4 & 2 & 1 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 0.75 & 0. & 0.25 \\ 0. & 1. & 0. \\ 0.25 & 0. & 0.75 \end{pmatrix} \quad A_3 = \begin{pmatrix} 0.375 & 0 & -0.125 \\ 0 & 0.5 & 0 \\ -0.125 & 0 & 0.375 \end{pmatrix},$$

calculer A_i^n, pour $n = 1, 2, 3 \ldots$ Que vaut $\lim_{n \to +\infty} A^n$?.

Exercice 8 On pose

$$A = \begin{pmatrix} 1 & -1 & 7 \\ -4 & 2 & 11 \\ 8 & 0 & 3 \end{pmatrix}, \; B = \begin{pmatrix} 3 & -2 & -1 \\ 7 & 8 & 6 \\ 5 & 1 & 3 \end{pmatrix}.$$

Que font les instructions suivantes

```
v=[1, 2, 4, 7];
A(v)=A(v) + 0.01;
B(v)=abs(B(v));
```

Exercice 9
1. Créer un vecteur de 11 coordonnées contenant les nombres $-5, -4, ..., 4, 5$.
2. Créer un vecteur de 1001 coordonnées contenant les nombres $-500, -499, -498..., 499, 500$. *linspace*
3. Créer un vecteur u contenant 10 valeurs entre 0 et π séparées par un incrément constant.
4. Créer un vecteur v tel que $v_{2i} = \cos u_{2i}$ et $v_{2i+1} = \sin u_{2i+1}$

Exercice 10 On rappelle deux approximations de la dérivée d'une fonction par différences finies

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}, \quad f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

1. Définir dans un M-file la variable $h = 0.1$ et un tableau d'abscisses x pour discrétiser finement l'intervalle $[0, \pi]$.
2. Calculer dans un tableau d la dérivée exacte de la fonction sin aux points x et dans un tableau $d1$ les valeurs approchées par la première formule ci-dessus. Calculer l'erreur maximale commise en faisant cette approximation sur l'intervalle $[0, \pi]$. Diminuer h, que remarque-t-on ?
3. Mêmes questions avec la deuxième approximation. Les valeurs approchées seront calculées dans un vecteur $d2$
4. Représenter sur le même graphique la dérivée et ses deux approximations

   ```
   plot(x,d,x,d1,x,d2)
   ```

   Que remarque-t-on, quelle est la meilleure approximation ?

3 Fonctions ou macros (`function`)

Nous allons maintenant détailler quelques règles de programmation qui pour la plupart ne devraient pas surprendre outre mesure si on connaît déjà un autre langage (C ou Fortran).
Les fonctions sont des enchaînements de commandes `Matlab` regroupées sous un nom de fonction permettant de commander leur exécution.
On peut mettre dans une fonction un groupe de commandes destiné à être exécuté plusieurs fois au cours du calcul avec éventuellement des valeurs de paramètres différents. La fonction peut aussi être chargée de réaliser un calcul avec un certain algorithme, qui pourra être remplacé éventuellement par un autre plus rapide ou plus précis, en changeant simplement le nom de la fonction dans le programme appelant. Enfin, dès que le programme est un peu long et compliqué, il est souhaitable de le découper en fonctions, correspondant à des étapes pour améliorer la lisibilité et la compréhension de l'algorithme.
Les fonctions peuvent être définies `inline` c'est à dire dans le corps du progamme appelant ou bien dans des fichiers externes indépendants `M_file`.

3.1 `M-files functions`

Dès que la fonction nécessite plusieurs instructions, il vaut mieux la définir dans un fichier à part à l'aide de l'éditeur de texte.
De manière générale, la syntaxe de définition d'une fonction externe est

```
function [y_1,...,y_m]=toto(x_1,....,x_n)
.
.
.
```

où `toto` est le nom de la fonction, $x_1, ...x_n$, les n arguments d'entrée et $[y_1, ...y_m]$ les m arguments de sortie. Les points verticaux symbolisent les instructions effectuées à l'appel de la fonction.
Le passage des arguments d'entrée dans les fonctions se fait par valeur. Aussi, même si elles sont modifiées dans la fonction les valeurs des paramètres ne sont pas modifiées dans le programme appelant.
Si une des variables de la procédure n'est pas définie à l'intérieur de celle-ci elle doit obligatoirement être fournie en argument d'entrée
La récupération des valeurs calculées par la fonction se fait par les paramètres de sortie ($[y_1, . * .y_n]$ dans la définition de `toto` ci-dessus).
Prenons l'exemple d'une fonction `angle`, qui doit calculer l'angle formé par le segment d'extrémités $(0, 0)$ et (x, y) avec l'horizontale et aussi le dessiner. On peut créer le fichier `angle.m` contenant les lignes :

```
function [s]=angle(x,y)
s=180*atan(y/x)/pi;
patch([x,0,x],[0,0,y],'y')
axis equal
```

puis dans la fenêtre de commandes on tape

```
angle(4,5)
```

Le nom du fichier contenant la fonction porte obligatoirement le nom de cette dernière. On peut mettre plusieurs fonctions dans le même `M-file` mais seule la fonction du même nom que le fichier peut être utilisée, *appelée*, à partir de la fenêtre de commandes ou d'une autre fonction ou d'un script. Les autres fonctions éventuellement stockées dans le fichier peuvent s'appeler entre elles mais ne sont pas visibles de l'extérieur.
S'il n'y a qu'un résultat comme dans l'exemple de la fonction `angle`, on peut se dispenser de le récupérer dans une variable. En revanche s'il y a plus d'un paramètre de sortie, il faut récupérer leurs valeurs dans des variables dans le script d'appel. Regardons par exemple l'utilisation de la fonction *polaire* définie ci-dessous et sauvée dans le fichier *polaire.m*

```
function [r,theta]=polaire(x,y)
r=sqrt(x^2+y^2);
theta=atan(y/x);
```

Pour l'utiliser à partir de la fenêtre `Matlab` on tape les instructions suivantes

```
>>r=polaire(2,3)    % ou bien
>>polaire(2,3)      % si seulement  le rayon nous intéresse
 ans  =
    3.6055513
>>[r,t]=polaire(2,3)  % si on veut récupérer à la fois le rayon et l'angle
 t  =
      .9828
 r  =
    3.6056
```

3.2 Inline functions

Une fonction ne comportant qu'un petit nombre d'instructions peut être définie directement dans la fenêtre de commandes de la manière suivante

```
>>angle=inline('180*atan(y/x)/pi')
angle =
     Inline function:
     angle(x,y) = atan(y/x)
>>angle(5,4)
 ans  =
    0.6747
```

Les arguments de la fonction `angle` sont normalement fournis à l'appel dans l'ordre d'apparition dans la définition de la fonction. On peut aussi spécifier les arguments d'appel explicitement

```
>>f = inline('sin(alpha*(x+y))','x','y','alpha')
f =
     Inline function:
     f(x,y,alpha) = sin(alpha*(x+y))
>>f(0.2,0.3,pi)
ans =
     1
```

3.3 Fonctions outils

Enfin pour clore ce paragraphe d'introduction à la programmation sous `Matlab`, notez que certaines commandes spéciales ne peuvent s'utiliser qu'en relation à une fonction :
`nargin`, donne le nombre d'arguments d'entrée passés à l'appel de la fonction.

```
function c = testarg1(a,b)
if (nargin == 1)
    c = 2*a;
elseif (nargin == 2)
    c = a + b;
end
```

`nargin` peut aussi être utilisée pour connaître le nombre prévu d'arguments d'entrée

```
>> nargin('testarg1')
ans =
     2
```

La commande `nargout` fonctionne de manière analogue pour les arguments de sortie.

Instruction	Description
`nargin`	nombre d'arguments d'entrée d'une fonction
`nargout`	nombre d'arguments de sortie d'une fonction
`error`	interrompt l'exécution de la fonction, affiche le message d'erreur et retourne dans le programme appelant.
`warning`	imprime le message mais ne retourne pas dans le programme appelant
`pause`	interrompt l'exécution jusqu'à ce que l'utilisateur tape un `return`
`pause(n)`	interrompt l'exécution pendant n secondes.
`pause off`	indique que les `pause` rencontrées ultérieurement doivent être ignorées, ce qui permet de faire tourner tous seuls des scripts requièrant normalement l'intervention de l'utilisateur.
`break`	sort d'une boucle `while` ou `for`.
`return`	retourne dans le programme appelant sans aller jusqu'à la fin de la fonction.

TAB. 2 – Commandes de contrôle.

3.4 Exercices sur les fonctions

find

Exercice 11 1. Écrire un *M_file* pour la fonction

$$f_1(x) = \frac{x^5 - 3}{\sqrt{x^2 + 1}}$$

2. Tester la fonction sur quelques valeurs, par exemple $f_1(1) = -1.4142$, $f_1(0) = -3$.
3. Créer un tableau x d'abscisses de -5 à 5 par pas de 0.1
4. Représenter la fonction f_1 aux points x_i, avec la commande

   ```
   plot(x,f1(x))
   ```

   Modifier si besoin `f1.m` pour que ceci soit possible.

Exercice 12 Reprendre l'exercice 10 et écrire une fonction recevant en argument d'entrée le paramètre h et renvoyant en paramètre de sortie l'erreur maximale entre la discrétisation avant et la valeur exacte de la dérivee de la fonction `sin`.

3.5 Algorithmes préprogrammés

Il existe de nombreux algorithmes préprogrammés dans `Matlab` pouvant être utilisés dans des programmes de simulation plus complexes comme “boîte noire”. Tous sont répertoriés et présentés dans l'aide en ligne. Leur nom peut se retrouver grâce aux menus `search` ou `index` de `help`).

A titre d'exemple nous détaillons ici la syntaxe pour utiliser la fonction `fzero`, pour trouver une racine d'une fonction d'une variable.
Syntaxe d'appel

```
x = fzero(fun,x0)
x = fzero(fun,x0,options)
```

Instruction	Description
fzero(f,a)	recherche des zéros d'une fonction f autour de a
quad(f,a,b)	calcul de l'intégrale d'une fonction f entre a et b
spline(xx,yy)	calcul de la spline cubique passant par les points (xx,yy)
fft(a)	transformation de Fourier rapide du vecteur a
ode23(f,t,y0)	résolution de l'équation y'=f(t,x), y(0)=y0

TAB. 3 – Quelques algorithmes préprogrammés

```
x = fzero(fun,x0,options,P1,P2,...)
[x,fval] = fzero(...)
[x,fval,exitflag] = fzero(...)
[x,fval,exitflag,output] = fzero(...)
```

Description
• `fun` est une fonction inline ou 'fun' pour une fonction `Matlab` ou @fun pour un M_file.
• `x = fzero(fun,x0)` trouve un zéro près de `x0`, si `x0` est un scalaire. La valeur `x` renvoyée par `fzero` est près d'un point où fun change de signe ou bien NaN si la recherche a échouée.
• Si `x0` est un vecteur à 2 composantes, `fzero` le comprend comme un intervalle tel que `fun(x0(1))` et `fun(x0(2))` sont de signes opposés. Si ce n'est pas le cas il y a une erreur.
• `x = fzero(fun,x0,[],P1,P2,...)` permet de passer des paramètres supplémentaires P1, P2, etc,... à la fonction fun.
Exemples :
1) Trouver un zéro d'un polynôme $p(x) = x^2 - x - 2$

```
p=inline('x^2-x-2')
x=fzero(p,-2.3)
x=fzero(p,2)
```

2) Trouver un zéro d'une fonction paramétrée

```
p=inline('x^2-a*x-2','x','a')  % définition de la fonction
a=1      % définition du paramètre
p(2,a)   % doit renvoyer 0
y=fzero(p,3,optimset('disp','off'),a)
```

Le 3ème argument de fzero indique qu'on ne veut pas afficher les messages éventuels. Le(s) 4ème (et suivant) argument(s) passe(nt) le(s) paramètre(s) supplémentaire(s) de la fonction (ici `a`). `p(y,a)` doit renvoyer 0.
Une description plus détaillée de la fonction `fzero`, mais en anglais... est bien sûr disponible dans l'aide en ligne.

4 Les boucles

Il y a deux types de boucles en `Matlab` : les boucles `while` et les boucles `for`. La boucle `for` parcourt un vecteur d'indices et effectue à chaque pas toutes les instructions délimitées par l'instruction `end`.

```
>>x=1; for k=1:4,x=x*k, end
x =
     1
x =
     2
x =
     6
```

```
x =
    24
```

la boucle `for` peut parcourir un vecteur (ou une matrice) en prenant comme valeur à chaque pas les éléments (ou les colonnes) successifs.

```
>>v=[-1 3 0]
v =
    -1     3     0
>>x=1; for k=v, x=x+k, end
x =
     0
x =
     3
x =
     3
```

La boucle `while` effectue une suite de commandes jusqu'à ce qu'une condition soit satisfaite.

```
>>x=1; while x<14,x=x+5,end
x =
     6
x =
    11
x =
    16
```

Les deux types de boucles peuvent être interrompus par l'instruction `break` . Dans les boucles imbriquées `break` n'interrompt que la boucle la plus interne.

Exercice 13
1. Définir dans un M_file un vecteur `H` contenant des valeurs croissantes entre 0 et 1.
2. Utiliser une boucle pour appeler la fonction écrite à l'exercice 12 avec l' argument d'entrée h prenant successivement les valeurs contenues dans `H`. Stocker l'erreur renvoyée par la fonction dans la composante correspondante d'un vecteur `E`.
3. Représenter graphiquement l'erreur en fonction du paramètre h à l'aide de la commande
   `plot(H,E)`
   Que remarque-t-on ?

5 Tests

Un test est une alternative entre deux commandes (ou groupe de commandes) `Matlab` sélectionnées suivant le résultat d'une opération logique. Le résultat d'une opération logique (voir tableau 4) est une variable logique ou boléenne qui vaut 1 pour `VRAI` et 0 pour `FAUX`.
Dans `Matlab` on dispose du classique `if-else` agrémenté du `elseif` parfois bien utile. La syntaxe est par exemple

```
>>x=16
 x  =
    16
>>if x>0, y=-x, else y=x, end
 y  =
  - 16
```

On a aussi la possibilité d'utiliser le `switch-case`, qui a son équivalent dans le langage C

```
switch expression
  case case_expr
    commande,... ,commande
  case case_expr1,case_expr2,case_expr3,...
     commande,... ,commande
...
  otherwise
    commande,... ,commande
end
```

Cet élément de syntaxe permet de gérer plus élégamment les tests portant sur un seul critère pouvant prendre plus de deux valeurs entières, comme dans l'exemple suivant

```
n=round(10*rand(1,1))
switch n
case 0
    fprintf('cas numero 0')
case 1
    fprintf('cas numero 1')
case 2
    fprintf('cas numero 2')
otherwise
   fprintf('autre cas')
end
```

français	test `Matlab`
et	&
ou	|
non	~
égal	==
différent	~=
plus petit que	<
plus grand que	>
plus petit ou égal à	<=
plus grand ou égal à	>=

Tab. 4 – Les opérateurs logiques dans les tests

On a vu dans le paragraphe 2.3 qu'on pouvait utiliser la plupart des fonctions définies pour un argument scalaire avec un argument tableau. Dans ce cas la fonction renvoie dans un tableau la valeur de la fonction appliquée à chaque composante. Si la fonction met en oeuvre des tests logiques sur la valeur de la variable scalaire passée en argument d'appel, le résultat appliqué à un vecteur dépend du résultat logique intersection du test appliqué à toutes les composantes. Pour appliquer la fonction composante par composante, il faut parfois la ré-écrire en remplaçant les tests sur des valeurs scalaires par l'emploi de la fonction `find`.
Prenons par exemple, la fonction suivante

```
function y=test1(x)
if x>0.5
  y=1
else
  y=0
end
```

qui teste la position d'une variable réelle par rapport à 0.5. Si on l'applique à un vecteur, on obtient une seule valeur correspondant au résultat du test scalaire vérifié pour toutes les composantes. Ici, `test1(v)` sera égal à 1 si toutes les composantes de v sont supérieures à 0.5.
Pour appliquer le test composante par composante, le plus simple est de remarquer que `test1` se programme aussi par l'instruction

```
y=(x>0.5)
```

qui donne le résultat attendu dans le cas scalaire et aussi vectoriel (le vérifier !)

Exercice 14 Regardons maintenant une deuxième fonction `test2`

```
function y=test2(x)
if x>0.5
  y=sin(x)
else
  y=x
end
```

1. Créer un vecteur contenant 10 valeurs aléatoires uniformément distribuées entre 0 et 0.5.
2. Tester l'emploi des deux fonctions `test1` et `test2` sur ce vecteur
3. Ecrire une fonction `test2v` qui fait la même chose que `test2` composante par composante. (Utiliser `find`)

Exercice 15

1. Ecrire un *M_file* pour la fonction

$$f_2(x) = \begin{cases} 0, & x < -1 \\ 1/(e^{-15(x+3/4)}+1), & -1 \leq x < -1/2 \\ 1, & -1/2 \leq x < 3 \\ e^{-15(x-7/2)^2}, & 3 \leq x < 4 \\ 0, & x \geq 4 \end{cases}$$

   sur l'intervalle $[-2, 5]$
2. Tester la fonction sur quelques valeurs, par exemple $f_2(-1) = 0.0230$, $f_2(3) = 0.0235$.
3. Créer un tableau d'abscisses x de -2 à 5 par pas de 0.01
4. Représenter la fonction f_2 aux points x_i, avec la commande

   ```
   plot(x,f2(x))
   ```

   Modifier si besoin `f2.m` pour que ceci soit possible.

6 Lecture et écriture au clavier et dans des fichiers

On peut avoir à lire des données numériques ou alphanumériques dans un fichier, par exemple les conditions initiales d'un calcul ou un maillage qui a été généré par un autre programme. Inversement, on peut vouloir sauver le résultat d'un calcul, pour s'en servir ultérieurement dans `Matlab` ou dans un autre programme.

```
M = dlmread('NomDeFichier','Delimiteur')
```

Lit des données numériques du fichier ASCII NomDeFichier, chaque nombre est séparé du suivant par le caractère Délimiteur ou par un retour à la ligne. La virgule et l'espace sont des délimiteurs par défaut.

```
[A,B,C,...] = textread('NomDeFichier','format')
```

Lit les variables avec le format spécifié jusqu'à épuisement du fichier. (Le nombre de variables à lire et le nombre de données dans le fichier doivent correspondre.)

```
fid=fopen('NomDeFichier')    % ouvre le fichier
A = fscanf(fid,format)
[A,count] = fscanf(fid,format,size)
```

Lit les données avec le format spécifié. Un `format` est une chaine de caractères spécifiant en lecture le type de données à lire :

```
'%d' pour un entier
'%f' pour un réel
'%c'pour un caractère.
```

On peut éventuellement spécifier le nombre maximal de données à lire avec le paramètre size, et le paramètre de sortie count renvoie le nombre de données effectivement lues.
Pour écrire des résultats sur l'écran la méthode la plus simple et de taper le nom de la variable sans la faire suivre d'un point virgule. Si on veut un format plus lisible, en particulier, afficher plusieurs variables sur la même ligne, on peut utiliser la commande `fprintf`, comme dans l'exemple ci-dessous

```
a=1.5;
b=2;
fprintf('a = %f et b= %d',a,b);
  a =1.5 et b=2
```

`dlmread('NomDeFichier','delimiteur')`	lecture du fichier
`dlmwrite('NomDeFichier',M,'delimiteur')`	écriture de M dans le fichier
`textread('NomDeFichier','format')`	lecture du fichier
`fid=open('NomDeFichier')`	ouverture du fichier NomDeFichier
`[A,count]=fscanf(fid,'format')`	lecture du fichier ouvert par open
`fprintf(fid,'format',données)`	écriture des données avec un format
`close(fid)`	fermeture
`fprintf('format',données)`	écriture des données avec un format
`fprintf('format',données)`	écriture des données avec un format

Tab. 5 – Les commandes d'entrée-sortie et les accès fichiers

7 Représentation graphique sous Matlab

Dans toutes les représentations graphiques, le logiciel se base sur des données discrètes rangées dans des matrices ou des vecteurs colonnes. Par exemple, pour représenter des courbes du type $y = f(x)$ ou des surfaces $z = f(x, y)$, les données `x,y,z` doivent être des vecteurs colonnes (`x` et `y`) ou des matrices (`z`) aux dimensions compatibles. L'instruction de dessin correspondante (par exemple `plot(x,y)` pour tracer des courbes planes) est alors utilisée et éventuellement complétée par des arguments optionnels (couleur, type de trait, échelle sur les axes, etc...). La visualisation du résultat s'effectue dans une fenêtre graphique (avec possibilité de zoom, de rotation, d'impression).

7.1 Exemple de représentation graphique en dimension deux

En exécutant le script suivant :

```
x=linspace(0,pi,30);   % crée un tableau de 30 composantes uniformément
% réparties entre 0 et pi
y=sin(x);
plot(x,y)    %relie les points (xi,yi) par un trait continu noir
plot(x,y,'p-b')  %relie les points (xi,yi) par un trait continu de couleur et
%matérialise les points avec un symbole
plot(x,y,'pb')  %matérialise les points (xi,yi) avec un symbole de couleur
```

Les points peuvent être matérialisés par le symbole p prenant les valeurs suivants : o . $*$ + $\times$
Les couleurs sont repérées par leur initiale en anglais : r(ed), b(lue), blac(k), w(hite), y(ellow), m(agenta), g(reen)
On peut rajouter un titre à la figure avec la commande `title`

```
title('sin(x) sur l''intervalle [0,pi]')
```

(Remarquer l'emploi d'une double apostrophe pour en faire figurer une dans une chaîne de caractères délimitée justement par deux apostrophes.)
On peut représenter plusieurs courbes sur la même figure de plusieurs manières : d'abord par un seul appel à la fonction `plot`

```
plot(x,cos(x),x,sin(x),x,exp(-x))    % Matlab va automatiquement utiliser des couleurs
%différentes pour chaque courbe
plot(x,cos(x),'o-r',x,sin(x),'x-b',x,exp(-x),'*-g')    % pour spécifier le type
% de symbole et la couleur à utiliser pour chaque courbe
legend('cos(x)','sin(x)','exp(-x)')  % pour rajouter une légende
```

Par défaut la fenêtre graphique est effacée avant chaque commande `plot`. Pour superposer des courbes par des appels successifs à cette fonction, il faut auparavant avoir utilisé la commande `hold on`. Par exemple le script suivant donnera le résultat représenté sur la figure 2

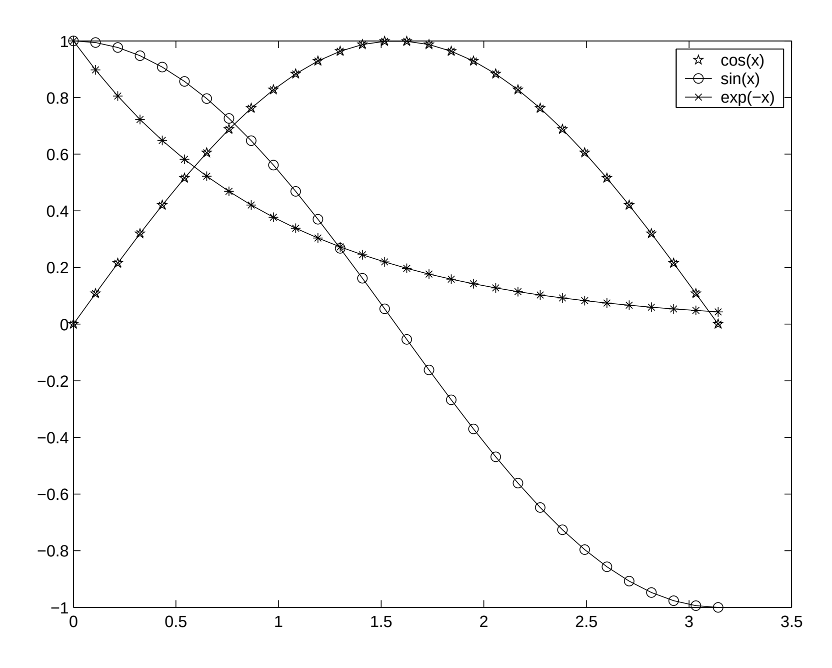


FIG. 2 – Exemple d'appels successifs à la fonction `plot`.

```
hold on
plot(x,cos(x),'o-r')
plot(x,sin(x),'x-b')
plot(x,exp(-x),'*-g')
legend('cos(x)','sin(x)','exp(-x)')
```

Instruction	Description
`plot(x,y)`	tracé de la courbe passant par les points `(x,y)`
`loglog(x,y)`	idem avec échelle logarithmique sur les deux axes
`semilogx(x,y)`	idem avec échelle logarithmique sur l'axe Ox
`semilogy(x,y)`	idem avec échelle logarithmique sur l'axe Oy
`plotyy(x,y,x,z)`	courbe (x,y) avec l'axe 0y à gauche, et courbe (x,z) avec l'axe 0z à droite
`xlabel('label')`	légende pour l'axe 0x
`ylabel('label')`	légende pour l'axe 0y
`title('label')`	titre au dessus du graphique
`legend('lab1','lab2','lab3',...)`	légende avec une chaîne de caractères pour chaque courbe
`text(x,y,'label')`	chaîne de caractères à la position x,y
`plot3(x,y,z)`	tracé de la surface passant par les points `(x,y,z)`
`hold on, hold off`	active/désactive la conservation de la fenêtre graphique à l'appel de la fonction plot

TAB. 6 – Principales instructions graphiques.

7.2 Autres types de représentation

Outre la représentation cartésienne de courbes ou de surfaces, il existe d'autres possibilités pour illustrer graphiquement un résultat. On peut citer parmi les plus utiles, les instructions `contour,ezmesh` (pour tracer les courbes de niveau d'une surface paramétrique), `mesh,ezplot3` (courbes paramétriques dans l'espace), `hist, rose` (histogramme d'un échantillon de données statistiques), etc...

8 Exercices d'analyse numérique

On présente maintenant sous forme d'exercices quelques exemples simples de simulations avec `Matlab`.

8.1 Interpolation

Exercice 16 *inline*

1. Définir dans un *M-file* la fonction
$$f(x) = \frac{1}{1+x^2}.$$
Définir un vecteur d'abscisses xx_i, avec $i = 1, 2, ...100$ uniformément reparties entre -5 et 5. Représenter graphiquement la fonction f aux abscisses xx par un trait continu vert. *spline*
2. Définir un vecteur de points d'interpolation x_i avec $i = 1, 2, ...10$, tels que $x_1 = -5 < x_2 < ... < x_9 < x_{10} = 5$. Calculer dans un tableau yy les valeurs aux abscisses xx de la spline cubique passant par les points x_i et interpolant la fonction f.
3. Représenter sur le même graphique la fonction f aux abscisses xx en vert, la spline aux abscisses xx en bleu et les points d'interpolation $(x, f(x))$ par des croix rouges.

Exercice 17 Afin d'illustrer graphiquement l'interpolation polynomiale de Lagrange, le script 2 calcule le polynôme d'interpolation d'une fonction quelconque associé à $(n+1)$ points équirépartis sur un intervalle avec la méthode des différences divisées. La fonction à interpoler est représentée en vert, le polynôme d'interpolation en bleu, et les points d'interpolation sont matérialisés avec des ronds rouges.

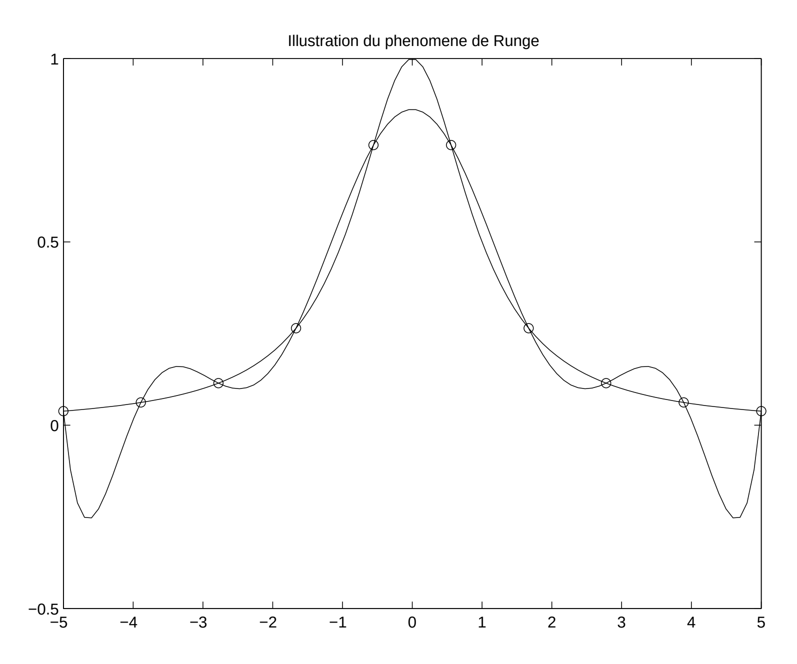


Fig. 3 – Interpolation de Lagrange de la fonction $1/(1+x^2)$ avec 10 points.

Script 2 *Interpolation polynomiale de Lagrange*

```
f=inline('sin(0.5*x)')
xmin=-5;xmax=5;
n=10;h=(xmax-xmin)/(n-1);
%interpolation de f(x)=sin(x/2) sur [-5,5]

xp=linspace(xmin,xmax,n);
d=f(xp);
 %calcul des différences divisées
for k=2:n
  for j=n:-1:k
    d(j)=(d(j)-d(j-1))/((k-1)*h);
  end
end
%estimation de Horner
x=linspace(xmin,xmax,100);
y=HHorner(d,xp,x);
plot(x,y,'-b',x,f(x),'-g',xp,f(xp),'or')
legend('polynome','fonction','points')
title('Interpolation de Lagrange')
```

Avec un M-file contenant la fonction d'évaluation du polynôme pour la représentation graphique.

```
function [y]=HHorner(c,x,t)
```

```
n=length(x);
y=c(n)*ones(size(t));
for i=n-1:-1:1
  y= y.*(t-x(i)*ones(size(t)))+c(i)*ones(size(t));
end
```

1. Vérifier le comportement de l'interpolation de Lagrange en choisissant comme fonction un polynôme.
2. L'algorithme est ensuite appliqué à la fonction $f(x) = \frac{1}{x^2+1}$ sur l'intervalle $[-5, 5]$. La représentation graphique conjointe de ce polynôme pour $n = 13$ et de la fonction est reproduite sur la figure 3 où le phénomène de non convergence (phénomène de Runge) est manifeste.

8.2 Résolution approchée des équations différentielles

On renvoie à [4] pour une présentation des équations différentielles et des méthodes numériques pour les résoudre.
`Matlab` possède plusieurs solveurs approchés pour les équations différentielles du type

$$\begin{aligned} y'(t) &= f(t, y(t)), \quad t \in \mathbb{R}, \\ y(t_0) &= y_0 \in \mathbb{R}^n. \end{aligned}$$

Le solveur choisi, par exemple `ode45, ode23, ode113, ode15s, ode23s, ode23t`, ou `ode23tb`, résout le problème de Cauchy ci-dessus en utilisant une méthode et une précision choisies par l'utilisateur. La syntaxe de base est

```
[T,Y] = solver(odefun,tspan,y0)
% odefun: fonction au second membre du systeme différentiel
%         ci dessus f(t,y)
% tspan : [t0,t1,...,tf] temps où on veut calculer la solution y
% y0    : Condition initiale y(t0) (vecteur colonne n composantes)
% T     : En sortie les temps où est calculée la solution
% Y     : Les valeurs de la solution aux temps T
```

Exercice 18 Dans cet exercice on s'intéresse à un modèle de dynamique des populations. On va illustrer graphiquement l'évolution du système proie-prédateur $(u(t), v(t))$ (u : prédateur et v : proie) suivant le modèle de Volterra :

$$(P) \quad \begin{cases} \frac{du}{dt}(t) = \frac{1}{2}u(t)(v(t) - 1), \quad t \in \mathbb{R}_+, \\ \frac{dv}{dt}(t) = v(t)(1 - u(t)), \\ u(0) = 2, \\ v(0) = 2 \end{cases}$$

1. On pose $Y = (u, v)^T$ la fonction vectorielle solution du système différentiel ci-dessus. Programmer dans un M-file la fonction $f(t, Y)$ de $\mathbb{R}^+ \times \mathbb{R}^2$ dans $\mathbb{R}^2$ de ce système. Attention ! le système est autonome (f ne dépend pas de t), mais `Matlab` s'attend à ce que la fonction passée en argument du solveur dépende de t et de Y.
2. Définir un vecteur T de 50 temps uniformément répartis entre 0 et 25. Calculer une solution approchée de (P) aux temps T en utilisant la fonction `Matlab ode23`
3. Représenter sur le même graphique l'évolution u en bleu et de v en rouge en fonction de t. Représenter dans une autre fenêtre l'évolution de v en fonction de u

On observera en particulier sur les figures 4 que le système semble périodique en temps.

Des explications plus détaillées sur ce modèle *proie - prédateur* sont disponibles dans un TP de maîtrise en SCILAB[6]

[6] `http://www.ann.jussieu.fr/~postel/scilab/volterra`

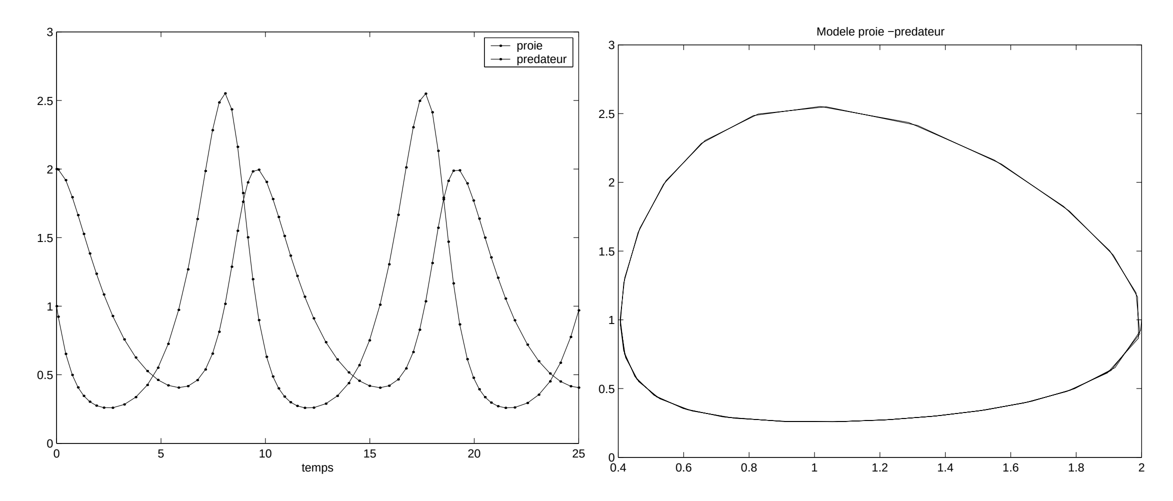


FIG. 4 – Modèle proie-prédateur, évolution des 2 populations en fonction du temps à gauche et d'une population en fonction de l'autre à droite.

Exercice 19 Dans cet exercice tiré de [5], on veut calculer la trajectoire d'un chien (C) lancé à la poursuite d'un lièvre (L). Le chien court à vitesse constante - le plus vite possible ! - mais en ajustant sa trajectoire de manière à ce que sa vitesse soit toujours colinéaire à la droite (LC). On note $Y_l(t) = (y_1^l(t), y_2^l(t))$ la trajectoire du lièvre, qui est une donnée du problème et $Y_c(t) = (y_1^c(t), y_2^c(t))$ celle du chien qui est l'inconnue

1. Traduire les hypothèses : le chien se déplace avec une vitesse constante en module v_c et la vitesse du chien colinéaire à la droite (LC).
2. Ecrire le système différentiel vérifié par $Y_c(t)$. Programmer en `Matlab` la fonction $f(t, y)$ de $\mathbb{R}^+ \times \mathbb{R}^2$ dans $\mathbb{R}^2$ de ce système différentiel.
3. Écrire un script pour intégrer $Y_c(t)$ à l'aide de `ode23`. On représentera sur le même graphique les trajectoires du lièvre et du chien en fonction du temps avec des couleurs différentes. Utiliser la fonction `Matlab pause` et des marqueurs pour visualiser l'avancement des deux protagonistes.
4. Tester avec différentes trajectoires pour le lièvre et différentes vitesses pour le chien.

Table des figures

Liste des tableaux

Références

[1] DANAILA, I., JOLY, P., KABER, S.-M., AND POSTEL, M. *Introduction à la modélisation et au calcul scientifique par la pratique.* Dunod, Paris, (2005).

[2] ALLAIRE, G., KABER, S.M. *Introduction à Scilab. Exercices pratiques corrigés d'algèbre linéaire.* Ellipses, Paris (2002).

[3] ALLAIRE, G., KABER, S.M. *Cours d'Algèbre Linéaire Théorique.* Ellipses, Paris (2002).

[4] CROUZEIX, M., MIGNOT, A.-L. *Analyse Numérique des Equations Différentielles.* Masson, Paris (1983).

[5] GANDER, W., HREBICEK, J. *Solving Problems in Scientific Computing Using Maple and* `Matlab` Springer-Verlag (1991)

[6] GUERRE-DELABRIERE, S., POSTEL, M. *Méthodes d'Approximattion - Equations Différentielles - Applications* `Scilab`. Ellipses, Paris (2004).

[7] LAPRESTÉ, J.-T. *Introduction à* `Matlab` Ellipses (2000)

Quelques sites internet

• le site officiel de `Matlab` www.mathworks.com[7]
• La version **html** de ce cours www.ann.jussieu.fr/~postel/matlab[8]
• Un cours d'analyse matricielle (niveau DEA) avec des applications `Matlab`
www.ann.jussieu.fr/~joly/matlab.php3[9]

Merci d'envoyer vos remarques et suggestions sur ce document par email à

`postel@ann.jussieu.fr`[10]

[7] `http://www.mathworks.com`
[8] `http://www.ann.jussieu.fr/~postel/matlab`
[9] `http://www.ann.jussieu.fr/~joly/matlab.php3`
[10] `mailto:postel@ann.jussieu.fr`